Algoritmos de Ordenação: QuickSort, MergeSort e BucketSort

Equipe: Fernanda Souza, Matias Benitez, Nikolas Jensen e Rafael Böeger

QuickSort

- Estratégia "*divide and conquer*":

1. Escolhemos um valor (designado *pivot*)
2. Partimos a sequência em duas parte:
    - todos os valores *menores do que o pivot* primeiro;
    - todos os valores *maiores do que o pivot* depois;
3. Recursivamente ordenamos as duas sub-sequências

Particionamento

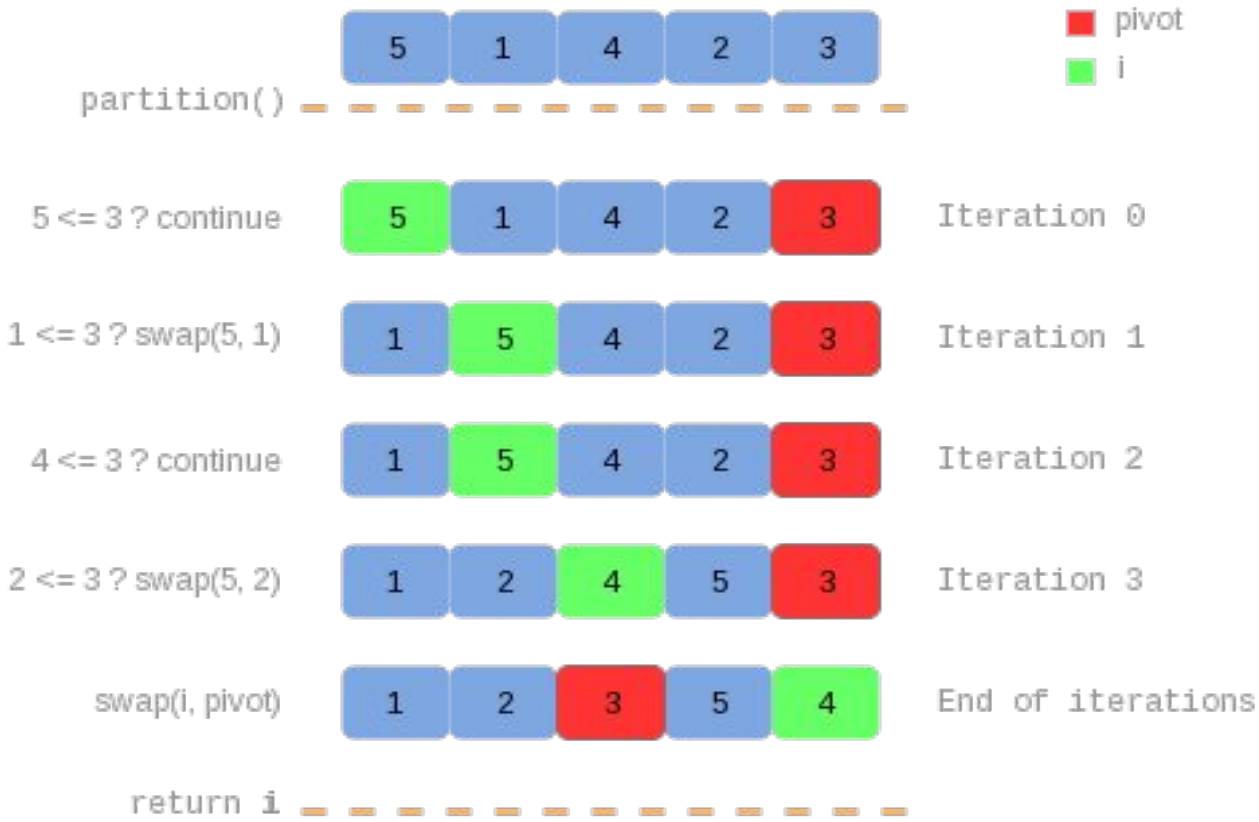
1. O método particionador deve primeiramente escolher um elemento que chamaremos de pivô. Em seguida iterar sobre toda a sequência a fim de posicionar todos elementos menores do que esse pivô à sua esquerda.

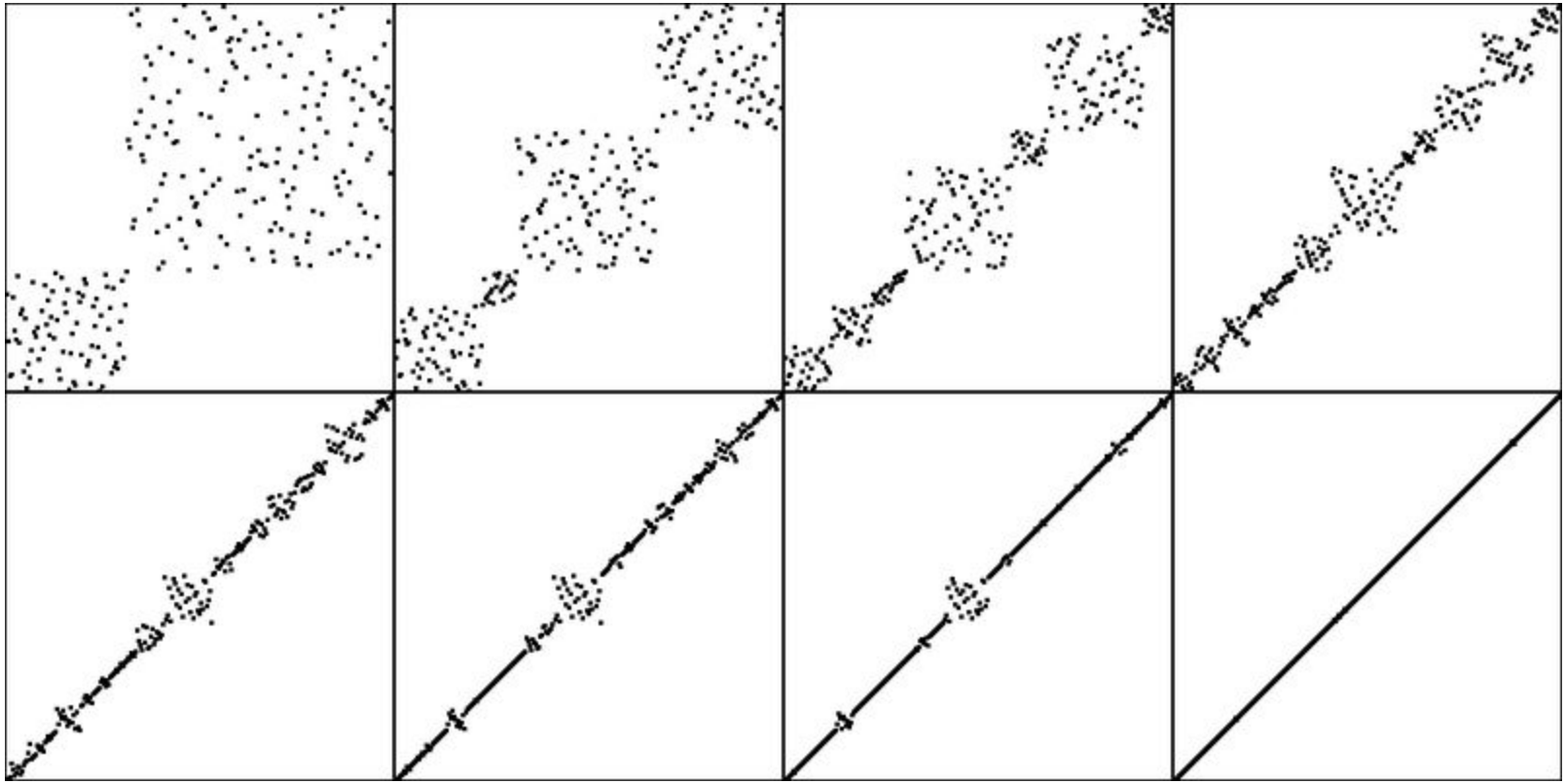
   Exemplo: <5, 1, 4, 2, 3>

2. O pivô dessa sequência será o número 3. Ao iterar, da esquerda para direita, em toda a sequência, toda vez que encontrarmos um elemento cujo valor seja menor do que nosso pivô, iremos colocá-lo à esquerda do pivô.
3. Para saber qual elemento já trocamos, devemos manter uma variável *i*, que nos permite controlar quais elementos já foram trocados, da esquerda para direita. Ao final de toda a iteração da sequência, pegamos nosso pivô (último elemento) e trocamos com a posição atual da variável *i*. Assim, temos a sequência particionada, porém ainda não completamente ordenada.

Recursão dentro do QuickSort

- Como ainda não se é garantido que os elementos a esquerda e a direita do pivô estejam ordenados, o algoritmo Quicksort utiliza o paradigma Dividir para conquistar para particionar o vetor recursivamente até que tenhamos subsequências ordenadas.
- Portanto, como primeiramente particionamos nosso vetor em duas partes e temos a posição do pivô, chamamos o método quicksort() recursivamente passando a metade à esquerda e em seguida a metade à direita.
- Ao final da execução, temos o vetor ordenado.


pivot
i
partition()
5 1 4 2 3
5 <= 3 ? continue
5 1 4 2 3
Iteration 0
1 <= 3 ? swap(5, 1)
1 5 4 2 3
Iteration 1
4 <= 3 ? continue
1 5 4 2 3
Iteration 2
2 <= 3 ? swap(5, 2)
1 2 4 5 3
Iteration 3
swap(i, pivot)
1 2 3 5 4
End of iterations
return i



```c
void LerSenhas(Linha **linhas)
{
    FILE *arq;
    int n = 0, tam, freq;
    char temp[250];
    arq = fopen("dados.txt","r");
  while (fscanf(arq, "%d %d %[^\n]", &tam, &freq, temp) == 3)
  {
        linhas[n] = (Linha*)malloc(sizeof(Linha));
        linhas[n]->tam = tam;
        linhas[n]->freq = freq;
        linhas[n]->palavra = (char*)malloc((tam+1) * sizeof(char));
        temp[tam] = '\0';
        strcpy(linhas[n]->palavra, temp);
        n++;
  }
  fclose(arq);
}
```

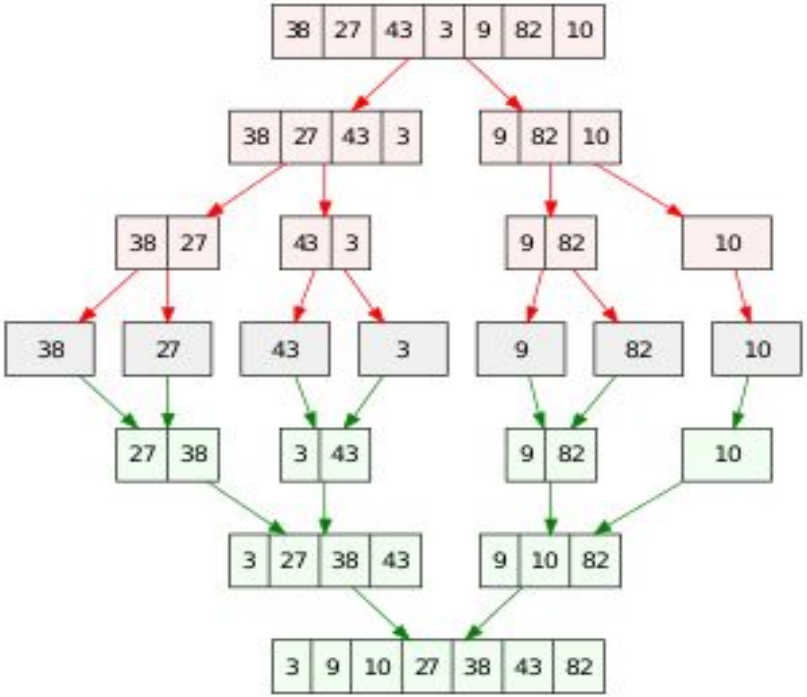
```c
int particionar(Linha **vet, int comeco, int final)
{
    Linha *pivo = vet[final];
    Linha *aux1;
    int i = comeco - 1;
    for(int j = comeco; j < final; j++)
    {
        if(strcmp(vet[j]->palavra, pivo->palavra) < 0)
        {
            i++;
            aux1 = vet[j];
            vet[j] = vet[i];
            vet[i] = aux1;
        }
    }
    if(strcmp(vet[final]->palavra, vet[i+1]->palavra) < 0)
    {
        aux1 = vet[final];
        vet[final] = vet[i + 1];
        vet[i + 1] = aux1;
    }
    return i+1;
}
```

```c
#define QTDSENHAS 430000

typedef struct linha{
  char *palavra;
  int tam;
  int freq;
} Linha;
```

```c
void QuickSort(Linha **vet, int comeco, int final)
{
    if(comeco < final)
    {
        int part = particionar(vet, comeco, final);
        QuickSort(vet, comeco, part - 1);
        QuickSort(vet, part + 1, final);
    }
}
```

MergeSort

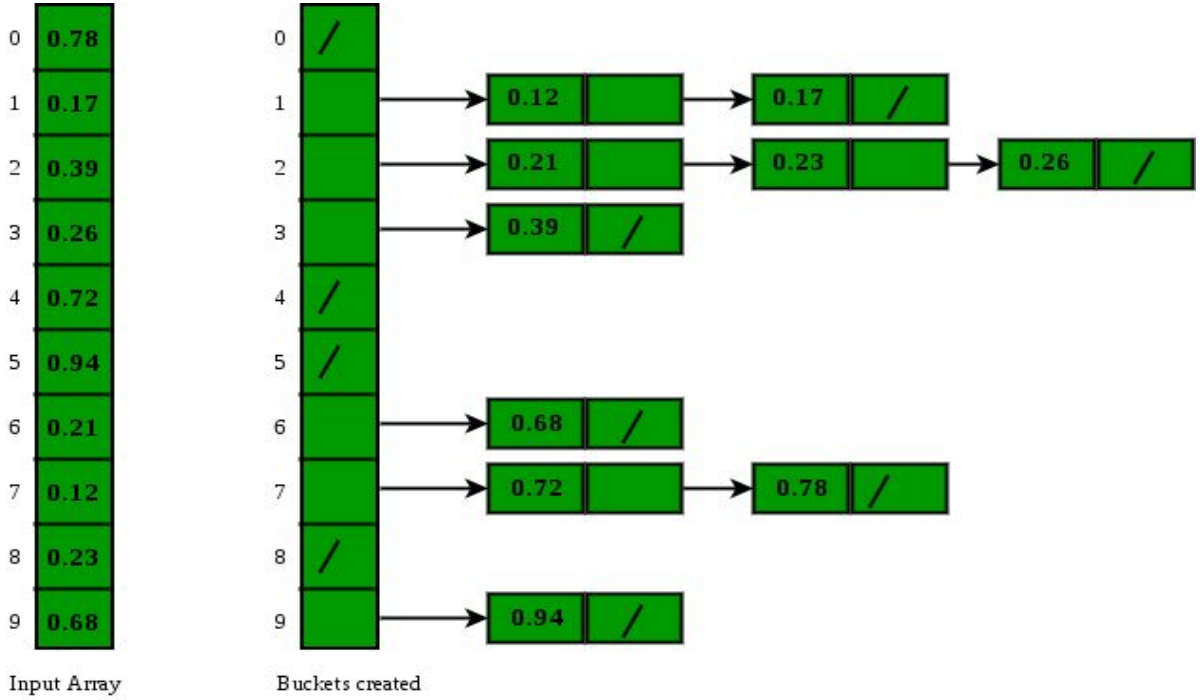

38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

```c
void Merge(Senha **senhas,int low,int mid,int high)
{
    int nL= mid-low+1;
    int nR= high-mid;
    Senha **L=malloc(sizeof(Senha *)*nL);
    Senha **R=malloc(sizeof(Senha *)*nR);
    int i;
    for(i=0;i<nL;i++)
    {
        L[i]=(Senha*)malloc(sizeof(Senha*[low+i]));
        L[i] = senhas[low+i];
    }
    for(i=0;i<nR;i++)
    {
        R[i]=(Senha*)malloc(sizeof(Senha*[mid+i+1]));
        R[i] = senhas[mid+i+1];
    }
    int j=0,k;
    i=0;
    k=low;
    while(i<nL&&j<nR)
    {
        if(strcmp(L[i]->palavra,R[j]->palavra)<0){
          senhas[k++] = L[i++];
        }
        else{
          senhas[k++] = R[j++];
        }
      }
        while(i<nL)
          senhas[k++] = L[i++];
        while(j<nR)
          senhas[k++] = R[j++];

    }
```

```c
void MergeSort(Senha **senhas,int low,int high)
{
    if(low<high)
    {
        int mid=(low+high)/2;
        MergeSort(senhas,low,mid);
        MergeSort(senhas,mid+1,high);
        Merge(senhas,low,mid,high);
    }
}
```

BucketSort


0
0.78
1
0.17
2
0.39
3
0.26
4
0.72
5
0.94
6
0.21
7
0.12
8
0.23
9
0.68
Input Array
0
1
2
3
4
5
6
7
8
9
0.12
0.17
0.21
0.23
0.26
0.39
0.68
0.72
0.78
0.94
Buckets created

```c
void bucketSort(Senha **array, int n, FILE *arq) {
    Bucket *buckets;
    char *inicio;
        size_t i;
    buckets = (Bucket *)malloc(QTD_LETRAS_ALFABETO * sizeof(Bucket));
    for (i = 0; i < QTD_LETRAS_ALFABETO; i++) {
        (*(buckets + i)).count = 0;
        (*(buckets + i)).value = NULL;  // necess·rio ?
    }
    for (i = 0; i < TAMANHO_VETOR; i++) {
        inicio = array[i]->palavra;
        size_t aux = *inicio - 97;
        insert_string_in_bucket_list(inicio, buckets + aux);
    }
    print_buckets(buckets, QTD_LETRAS_ALFABETO, arq);
}
```

```c
void insert_string_in_bucket_list(char *string, Bucket *bucket) {
    if (bucket->count == 0) {
        bucket->value = (char **)malloc(sizeof(char *));
    } else {
        bucket->value = (char **)realloc(bucket->value, sizeof(char *) * (bucket->count + 1));
    }
    *(bucket->value + bucket->count) = (char *)malloc(sizeof(char) * strlen(string));
    memcpy(*(bucket->value + (bucket->count)++), string, strlen(string));
}

```

Tabela de Tempos para comparação

Algoritmo	Máquina 1		Máquina 2		Máquina 3	
MergeSort	11,102	11,155	11,045	11,068	12,098	12,136
QuickSort	0,245	0,270	0,157	0,166	0,358	0,358
BucketSort	0,154	0,160	0,131	0,132	0,231	0,244
BucketSort + BubbleSort	58,092	57,954	57,920	57,093	63,037	64,901

Tabela de Tempos médios para comparação

Algoritmo	Máquina 1	Máquina 2	Máquina 3	Média
MergeSort	11,116	11,057	12,017	11,356
QuickSort	0,259	0,162	0,358	0,260
BucketSort	0,157	0,131	0,237	0,175
BucketSort + BubbleSort	58,023	57,506	63,969	59,832